2012 28th IEEE International Conference on Software Maintenance (ICSM)

Detecting Similar Applications with Collaborative Tagging

Ferdian Thung, David Lo, and Lingxiao Jiang
School of Information Systems
Singapore Management University, Singapore
{*ferdianthung,davidlo,lxjiang*}*@smu.edu.sg*

***Abstract*—Detecting similar applications are useful for various purposes ranging from program comprehension, rapid prototyping, plagiarism detection, and many more. McMillan et al. have proposed a solution to detect similar applications based on common Java API usage patterns. Recently, collaborative tagging has impacted software development practices. Various sites allow users to give various tags to software systems. In this study, we would like to complement the study by McMillan et al. by leveraging another source of information aside from API usage patterns, namely software tags. We have performed a user study involving several participants and the results show that collaborative tagging is a promising source of information useful for detecting similar software applications.**

I. Introduction

Search engines like Google allow users to find similar webpages. Code search engines, such as JavaClan proposed by McMillan et al. [12], can detect applications of similar functionalities. Detecting similar applications has various benefits for program comprehension, rapid prototyping, plagiarism detection, and many more [8], [10], [14], [16], [18].

McMillan et al. investigate how Java API methods are called within an application. These API method calls are referred to as semantic anchors and are used as features to detect similar applications. In this work, we would like to complement their study by considering another source of information that are often available on various project hosting platforms, such as SourceForge[1].

Recently, collaborative tagging has been a popular way to crowdsource and share knowledge. Tagging has also been used in software engineering communities. Various project hosting sites like SourceForge allow people to provide tags to various software systems. Since similar applications are likely to be tagged in a similar way, these tags can provide information useful for detecting similar applications.

In this paper, we propose to leverage tags for detecting similar application. We first collect more than a hundred thousands of projects from SourceForge, and extract tags for the projects. Then, we discriminate important tags from less important ones. Some tags do not carry much information; for example, many software systems are tagged with "English" as they are released in English, but they could potentially be very different from one another. Other tags, such as "Word Processor", carry more information, and applications with this tag are more likely to share similar core functionality. Thus, to detect similar applications, we infer different weights for different tags. After tag weights are learned, each application could be represented by a set of tags with weights. The similarity between two applications can then be measured by comparing their representative tags.

We compare our solution that leverages tags with the approach by McMillan et al. (named JavaClan) that leverages Java API method calls within application code based on a corpus of more than a hundred thousands of applications from SourceForge. Given an application used as a query, our solution and JavaClan would recommend 10 most similar applications respectively. We perform a user study by asking several participants to indicate if they agree whether each recommended application has similar functionality to the query application. Then, we measure the effectiveness of our approach and JavaClan by using the following measures: *success rate*, *confidence* [12], and *precision* [12]. Based on 20 queries used in the user study, our approach could achieve a higher success rate, higher average confidence, and higher average precision than JavaClan: Our approach achieves a *success rate* upto 80%, while JavaClan achieves upto 65%; our approach achieve higher confidences and precisions than JavaClan for about 65% and 35% of the query tasks, respectively; For another 15% and 40% of the query tasks, our approach achieves the same confidences and precisions as JavaClan. In addition, we find that many software systems that we recommend are different from those of JavaClan. Thus, the two approaches are complementary and could potentially be combined in the future.

Our contributions are as follows:

1) We propose a new solution to detect similar applications by leveraging collaborative tagging. This is complementary with the state-of-the-art work by McMillan et al. which only utilizes Java API calls to detect similar applications. Different from the previous approach that is language specific, our approach can detect similar applications in different programming languages.
2) With a user study, we compare our proposed approach with JavaClan and show that our approach could outperform the previous approach.

The structure of this paper is as follows. In Section II, we describe preliminary information about tagging on Source-

[1]http://sourceforge.net/

978-1-4673-2312-3/12/$31.00

Figure 1. Sample Tags from SourceForge

Forge. In Section III, we elaborate our proposed approach. We present our empirical evaluation in Section IV. We discuss related work in Section V. Finally, we conclude and mention future work in Section VI.

II. Tagging in SourceForge

SourceForge is an established and well-known site that aggregates various software systems for users to download. It allows people to provide not only the descriptions of an application but also tags for the application. Tags at times are more useful than textual descriptions as they often capture the key characteristics or features of an application.

SourceForge allows various kinds of tags to be specified. A snapshot of tags on SourceForge is shown in Figure 1. The tags describe the categories a software system may belong to (in this case CMS System, and Site Management), the license of the software system, the languages in which the system is available, the intended audience, the type of user interface, and the programming languages.

Some of these tags are more useful than others in inferring similar applications. Categories are possibly the strongest indicator, while languages (especially common languages like English) are possibly the weakest. Nevertheless, we collect all of these tags and use them in our study.

III. Proposed Approach

Our approach is illustrated in Figure 2. It mainly consists of three steps: data collection, weight inference, and similar application detection.

Data Collection: We first create a pool of applications from which similar applications could be found. We collect

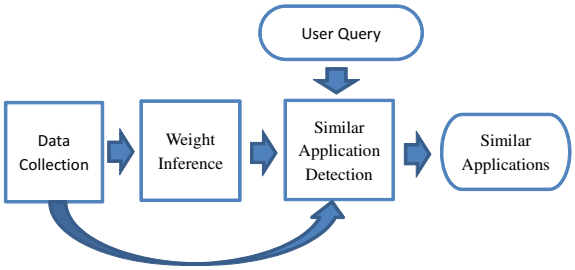

Figure 2. Proposed Approach

applications from SourceForge which provides APIs[2] that allow us to find various information about the applications hosted there. The APIs return information in the JSON format. We process the information for more than a hundred thousand applications and store them in our own database.

Weight Inference: Some tags are more important for determining the similarities of two applications than other tags. Tags that are shared by all or a majority of applications are not very useful—otherwise all applications would be deemed as similar. On the other hand, tags that are shared by a small groups of applications are more important.

With this intuition, we weight each tag based on the number of applications that are tagged by it. The more applications a tag tags, the less its weight is. This follows the concept of inverse document frequency in information retrieval [11]. Let $App(t)$ denote the set of applications that are tagged by t and $|App(t)|$ be the size. The weight of t, denoted as $w(t)$, is given as: $w(t) = \frac{1}{|App(t)|}$.

Similar Application Retrieval: We characterize each application by the set of its tags. To find similar applications, we compare the sets of tags that two applications posses. Let $Tag(A)$ denotes the set of tags for the application A. The similarity of two applications A_1 and A_2 are given as follows, which is based on cosine similarity [20]:

$$sim(A_1, A_2) = \frac{\sum_{t \in Tag(A_1) \bigcap Tag(A_2)} w(t) \times w(t)}{\sqrt{\sum_{t \in Tag(A_1)} w(t)^2} \times \sqrt{\sum_{t \in Tag(A_2)} w(t)^2}}$$

The above formula would assign a higher score if two applications share more tags and the tags have higher weights. The denominator of the formula is used to normalize the score. If an application has many tags, it is more likely that it coincidentally shares a tag with another application.

Given an application as a query, we compare it with each of other applications in our database. We only keep the top-10 similar applications and incrementally update the top-10 results as we scan through all applications. We find this search technique is sufficiently fast—we can detect similar applications in less than two seconds in a database containing more than a hundred thousand applications.

IV. Empirical Evaluation

In this section, we describe our dataset, experimental settings, research questions, and answers to these questions.

[2]http://sourceforge.net/apps/trac/sourceforge/wiki/API

A. Dataset & Experimental Settings

We retrieve 164,535 projects from SourceForge with its APIs. These projects form the pool of applications that we query for similar applications.

We make use of 20 queries listed in Table I used by McMillan et al. [12]. They investigate 24 unique queries to evaluate JavaClan,[3] but we only pick a subset. This subset is the set of queries for which *all* of the top-10 recommended applications made by JavaClan still exist on SourceForge.[4]

Table I
QUERIES

No	Query	No	Query	No	Query
1	bcel	8	drawswf	15	qform
2	bigzip	9	genesys-mw	16	redpos
3	certforge	10	javum	17	sqlshell
4	chiselgroup	11	jazilla	18	tyrex
5	classgen	12	jsresources	19	xflows
6	color-studio	13	opensymphony	20	yapoolman
7	confab	14	psychopath		

In the study by McMillan et al., most applications are investigated by one participant. Participants give a score from 1–4 to indicate their confidences in whether two applications are similar. We modify the scale to 1–5 by following the commonly used Likert scale [1]. The rating items are: 1. strongly disagree (i.e., the two applications are very dissimilar), 2. disagree, 3. neither agree or disagree, 4. agree, and 5. strongly agree (i.e., the two applications are very similar). We perform a user study to compare our approach with JavaClan.

For each of the 20 queries used in our study, we take the top-10 recommendations given by JavaClan (retrieved by their search engine[5]) and the top-10 recommendations given by our approach. Thus, we have 20 recommendations for each query, possibly including duplicates. We randomly mix and shuffle these recommendations without duplicates, and for each of the recommendations we ask a participant the question: Is this recommended application similar to the query application? To decide the rating for the questions, the participants are allowed to browse the websites that provide information about the applications. The random shuffling is important to reduce experimental bias toward either of the approaches. All of the (at most) 20 questions for each query are assigned to the *same* person to reduce opinion differences among different persons.

With the ratings for all of the questions for all queries, we measure the effectiveness of our approach and JavaClan by using three metrics: success rate, confidence, and precision, the later two have been used to evaluate JavaClan [12]. We introduce the definitions of these metrics as follows:

A search result is *successful* if at least one application in the top-10 list is rated 3 or above. The *success rate* is defined as the proportion of queries for which the search result is successful. The *confidence* of a participant for a question is the Likert rating given by the participant for the question. The *average confidence* is the average rating across all questions by all participants. The *precision* for each query is the proportion of applications in the top-10 list that are labeled by a participant as similar (i.e., 4 or 5 in our Likert scale). The *average precision* across all queries is also used as an effectiveness metric.

B. Research Questions

We investigate the following research questions:

- RQ1 How many queries could our proposed approach and JavaClan return successful search results?
- RQ2 How high is the average confidence of the participants when using our approach as compared with JavaClan? Is our result statistically significantly better than that of JavaClan?
- RQ3 How high is the average precision of our approach and that of JavaClan? Is our result statistically significantly better than that of JavaClan?

C. RQ1: Success Rate

The success rate of our approach and JavaClan is shown in Table II. We note that our approach leads to more successful search results than JavaClan. When we further strengthen the requirement of success from "rated 3 or above" to "rated 4 or above", our approach still perform better than JavaClan, although the success rates for both approaches are lower (Column "Success Rate (Above 4)").

Table II
OUR APPROACH VS. JAVACLAN: SUCCESS RATE

Approach	#. of Successful Queries	Success Rate (Out of 20)	Success Rate (Above 4)
Ours	16	80%	50%
JavaClan	13	65%	35%

D. RQ2: Confidence

Table III shows various statistics of the confidences that participants have on the results of our approach and those of JavaClan. We note that the similar applications recommended by our approach have better median and arithmetic mean confidences than JavaClan. We also compute the average confidence *for each query*. Out of 20 queries, our average confidences are higher in 13 queries than JavaClan and the same in 5 queries.

Table III
OUR APPROACH VS. JAVACLAN: CONFIDENCE

Approach	Sample Size	Min	Max	Median	Mean
Ours	200	1	5	2	2.02
JavaClan	200	1	5	1	1.715

We have also performed Mann-Whitney U test (a non-parametric test of statistical significance) at 0.01 confidence level[6]. We find that the value for the test is 0.001. This means that the difference is statistically significant.

[3] cf. questionnaires at http://www.cs.wm.edu/semeru/clan/#experiment.

[4] Many applications are ephemeral on SourceForge.

[5] http://javaclan.net

[6] We do not perform a statistical test on success rates in RQ1 since they are just two numbers.

E. RQ3: Precision

Table IV shows various statistics about the precisions of the results from our approach and those from JavaClan. We note that, in both approach, the minimum and maximum precisions are the same, while our approach have higher median and arithmetic mean precisions. This indicates that our approach gives an overall better performance than JavaClan. In addition, we compute the average precision *for each query*; Out of 20 queries, our mean precisions are higher in 7 queries than JavaClan and the same in 8 queries.

Table IV
OUR APPROACH VS. JAVACLAN: PRECISION

Approach	Sample Size	Min	Max	Median	Mean
Ours	20	0	0.4	0.05	0.115
JavaClan	20	0	0.4	0	0.095

We have also performed Mann-Whitney U test at 0.01 confidence level and find that the test value is 0.488. This means that the difference is *not* statistically significant.

F. Threats to Validity

We have only three participants in this user study. We have also only investigated 20 queries. In the future, we plan to reduce this threat to validity by increasing the number of participants and the number of queries.

V. RELATED WORK

The closest to our work is the study by McMillan et al. [12]. Our approach is different as we consider a complementary source of information, collaborative tagging, for detecting similar applications. Our approach can work on applications in different programming languages. The semantic tags given by end users are likely more relevant than information inferred from API method calls. However, not all applications have sufficiently relevant tags. In the future, we plan to combine both approaches together to improve search confidence and precision.

There are also a number of studies that propose various code search engines for returning similar code pieces, functions, components, applications, etc., such as Exemplar [5], Mica [19], Portfolio [13], SNIFF [3], Sourcerer [9], and XSnippet [17]. However, to the best of our knowledge, we are the first to use collaborative tags to detect similar applications, which are complementary to other approaches.

Many studies also aim to detect similar code fragments (a.k.a. clone detection) based on text matching, syntax trees, program dependence graphs, etc. [2], [4], [6], [7], [15]. Different from these studies, we recover similar applications rather than code. Two applications can implement the same functionality but are written in different ways. These two would be similar applications but not clones of each other.

VI. CONCLUSION AND FUTURE WORK

We have proposed an approach that detects applications of similar functionalities by leveraging collaborative tagging. This complements the recent study by McMillan et al. that uses API calls for detection. Applications from many project hosting platforms, such as SourceForge, have user-given tags that could shed light on the nature of the applications. We collect such tags from SourceForge and perform weight inference to detect similar applications. We have evaluated the effectiveness of our approach with a user study and three metrics: success rate, confidence, and precision, and show that our approach can achieve better results than JavaClan.

There is still room for improvement, considering that both JavaClan and our approach have relatively low confidences and precisions. For example, it may be useful to explore the semantic aspects of the tags, such as synonyms, to improve our approach. It is also interesting to note that many search results from our approach are different from those of JavaClan. In the future, we plan to combine various approaches to enhance the search of similar applications.

REFERENCES

[1] E. Allen and C. Seaman, "Likert scales and data analyses," *Quality Progress*, vol. July, 2007.
[2] B. S. Baker, "On finding duplication and near-duplication in large software systems," in *WCRE*, 1995.
[3] S. Chatterjee, S. Juvekar, and K. Sen, "SNIFF: A search engine for java using free-form queries," in *FASE*, 2009, pp. 385–400.
[4] M. Gabel, L. Jiang, and Z. Su, "Scalable detection of semantic clones," in *ICSE*, 2008, pp. 321–330.
[5] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *ICSE (1)*, 2010, pp. 475–484.
[6] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in *ICSE*, 2007.
[7] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE TSE*, vol. 28, no. 7, pp. 654–670, Jul 2002.
[8] K. Kontogiannis, "Program representation and behavioural matching for localizing similar code fragments," in *CASCON*, 1993.
[9] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and searching internet-scale software repositories," *Data Min. Knowl. Discov.*, vol. 18, no. 2, pp. 300–336, 2009.
[10] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: Detection of software plagiarism by program dependence graph analysis," in *KDD*, 2006.
[11] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[12] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *ICSE*, 2012.
[13] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *ICSE*, 2011.
[14] A. Michail and D. Notkin, "Assessing software libraries by browsing similar classes, functions and relationships," in *ICSE*, 1999.
[15] N. H. Pham, H. A. Nguyen, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Complete and accurate clone detection in graph-based models," in *ICSE*, 2009, pp. 276–286.
[16] T. Sager, A. Bernstein, M. Pinzger, and C. Kiefer., "Detecting similar java classes using tree algorithms," in *MSR*, 2006.
[17] N. Sahavechaphan and K. Claypool, "XSnippet: mining for sample code," in *OOPSLA*, 2006, pp. 413–430.
[18] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for java," in *ASE*, 2007.
[19] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding api components and examples," in *VLHCC*, 2006, pp. 195–202.
[20] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.